MEMSET

Using memset to scrub sensitive data in memory does not usually work unless the data is used subsequently.

Sean Barnum, Cigital, Inc. [vita[1]]

2007-03-29

Part "Original Cigital Coding Rule in XML"

Mime-type: text/xml, size: 5445 bytes

Attack Category	• Memory Scanning
Vulnerability Category	• Compiler Optimization • Information Leakage
Software Context	• Memory Management • Cryptography
Location	
Description	Using memset to scrub sensitive data in memory (where the memory is subject to dead-store removal optimization) does not usually work unless the data is used subsequently. The memset() function sets the bytes in a block of memory to the specified value. It may seem sensible to use this to zero out memory containing sensitive data after that data is no longer needed. However, most compilers will simply omit the memset call from object code as an "optimization." If the memory is zeroized and is never accessed afterwards, the compiler sees it as an unnecessary assignment. This is problematic for data such as cryptographic keys that need to be removed from memory after use.
APIs	**Function Name** / **Comments** memset
Method of Attack	
Exception Criteria	The false positive rate can potentially be limited through the following: x=malloc(z) memset(x,0,z) Where the memset immediately follows a malloc, the memory should contain no sensitive data. Therefore, the memset will not usually be optimized away (unless the memory was pointlessly allocated,

1. http://buildsecurityin.us-cert.gov/bsi-rules/35-BSI.html (Barnum, Sean)

	it is reasonable to expect that it will eventually be read or written to after the memset call).
Solutions	

Solution Applicability	Solution Description	Solution Efficacy
All platforms.	There are three ways to avoid having the memset call optimized out by the compiler. The first is to access the buffer again after the memset call in a way that would force the compiler not to optimize the location. This can be achieved by memset(buffer, 0, sizeof(buffer)); *(volatile char*)buffer = *(volatile char*)buffer;	Solution should be effective on most platforms, but consult platform documentation to verify that it suffices to reference one character in this fashion.
Windows and other platforms that provide a secure method for zeroing memory.	A second solution is to replace the memset call with other code that will not be optimized out. Newer versions of the Platform SDK on Windows include a function called SecureZeroMemory that will zero out a buffer in a way that will not be optimized.	Effective. Preferred solution, when available.
Windows and other platforms that provide	A third solution is to turn off optimizations	Effective.

	a means of controlling compiler optimization.	for the code in question. With the Microsoft cl compiler you can do this by placing the following pragma at the top of the source file: #pragma optimize("g", off) Or you could surround the function calls in question with #pragma optimize("", off) and #pragma optimize("", on) to turn off optimization for a particular section of code.	
Signature Details	void *memset(void * buffer, int c, size_t num)		

Examples of Incorrect Code	`[...]`
	`memset(key, 0, 32);`
	`[...]`		

Examples of Corrected Code	`[...]`
	`void *secureMemset(void *v,int c,size_t n) {`
	`volatile char *p = v;`
	`while (n--)`
	`*p++ = c;`
	`return v;`
	`}`
	`[...]`
	`secureMemset(key, 0, 32);`
	`[...]`		

Source Reference	• Howard, Michael. "Some Bad News and Some Good News[2]" (MSDN Library article).		
Recommended Resource			
Discriminant Set	**Operating Systems**	• UNIX	

	• Windows
Language	

Cigital, Inc. Copyright

The Build Security In (BSI) portal is sponsored by the U.S. Department of Homeland Security (DHS), National Cyber Security Division. The Software Engineering Institute (SEI) develops and operates BSI. DHS funding supports the publishing of all site content.

1. mailto:copyright@cigital.com